USING DOMAIN-SPECIFIC MODELING AND MODEL DRIVEN DEVELOPMENT TO DEVELOP SOFTWARE DEFINED RADIO COMPONENTS AND APPLICATIONS

Dominick Paniscotti
(PrismTech,
Saddle Brook, NJ, 07633 USA
dp@prismtech.com)

Bruce Trask
(PrismTech,
Saddle Brook, NJ, 07633 USA
bt@prismtech.com)

Angel Roman
(PrismTech,
Saddle Brook, NJ, 07633 USA
ar@prismtech.com)

Vikram Bhanot
(PrismTech
Saddle Brook , NJ, 07633 USA
vb@prismtech.com)

ABSTRACT

Software Defined Radios are in a unique software domain within DRE systems. This domain includes tackling complexities introduced from a number of quarters, including concerns dealing with systems that are embedded, real-time, high performance, heterogeneous, platform-independent, portable, object-oriented, multi-threaded, distributed, component-based, programmable via both declarative and imperative programming mechanisms, and in many cases resource constrained. Needless to say, domains as complicated as this stand to gain from the recent advances in Model Driven Development, Generative Programming and Agile Programming techniques. Just as the domain is multi-faceted in nature so to need be the solutions, tools and techniques applied to them to ensure success.

This paper will describe the application of these solutions, tools and techniques to the Software Defined Radio domain. It will include detailed descriptions of not only how Model Driven Development techniques and tools can be used but also descriptions of Generative Programming techniques that can be successful leveraged including domain-specific languages as applied to the SCA and the Software Radio Domain.

1. GENERAL BACKGROUND

For the past twenty years, there has been a continuous evolution in electronic communications equipment. The evolution can be described as one of moving the radio functionality from being located in the hardware platform running with proprietary processors and circuitry to being located in firmware running on programmable logic and then to being located in software running on general purpose processors. The driving force behind this evolution has been the need to leverage the inherent greater malleability and configurability of software versus that of hardware. As radio functionality continues to move into software, or looking at it another way, as that software moves “closer to the antenna”, it becomes more commercially viable to maintain, configure, test and reuse communications algorithms and functionality as well as the hardware on which it runs. This evolution is very similar to that of the computer itself with today's PCs running applications, the bulk of which exist as software running on general purpose hardware.

The communications industry has coined a term for this type of communications equipment: the Software Defined Radio[14].

As the radio and communications domain moves into a software centric solution, it is only natural that it leverage advances in the software domain as part of its implementation. These advances include object orientation, frameworks, component based design, middleware, in addition to imperative and declarative languages. More recently, the rise in abstraction level of the radio platform in the form of operating systems and middleware in combination with advances in modeling tools has opened the door to allow the evolution of communications software to enter the realm of model-driven development. This is fortuitous as the complexity of these communications systems has increased so dramatically that the viability of these new systems now hinges on the increased productivity, correctness and robustness that model-driven development affords.

This paper details the application of model-driven development, and more specifically, domain-specific modeling to the software defined radio domain. This domain has very unique characteristics as its systems typically are a confluence of a number of typically challenging aspects of software development. To name a few, these systems are usually described by modifiers such as, embedded, real-time, distributed, object-oriented, portable, heterogeneous, multithreaded, high performance, dynamic, resource-constrained, safety-critical, secure, networked, component based and fault-tolerant. Each one of these modifiers by themselves carries with it a set of unique challenges but building systems characterized by all of these modifiers all at the same time makes for quite an adventure in software development. In addition to all of these, it is quite common in these embedded systems for components to have multiple implementations that must run on disparate processing elements. With all of this taken into account, it stands to reason that these systems could and should benefit greatly from advances in software technology such as domain-specific modeling and model-driven development.

2. DETAILED BACKGROUND

In 1999, a consortium of the leading U.S. military radio developer companies created the Software Communications Architecture (SCA)[1]. This is one of many possible software defined radio models that can be input into the Domain Specific Modeling process and techniques discussed in this paper. We chose this one as concrete example of one such architecture since it is an open standard freely available to all.

This SCA defines five primary aspects of next-generation communications equipment software

- A standard component object model
- A standard deployment and configuration framework
- A standard declarative programming format for describing software components and how they are connected together
- A standard portability layer upon which component run
- A standard messaging format for intercomponent communication

As a result, the SCA significantly furthers standardization of the software radio domain and thus brings many benefits to the domain such as interoperability, portability, reuse, and a level of architecture consistency. However, the SCA specification does not solve all of the issues associated with implementing these complex systems. Some of the problems that remain include:

- Labor intensive implementations of the SCA object model in 3GL languages
- Lack of architectural consistency at various levels of implementations
- The learning curve of the specification and lack of effective training materials
- The technology gaps between software developers and radio domain experts
- Ensuring correctness of implemented systems
- The dynamic nature of the SCA, which opens the door to a host of runtime errors that would best be “left shifted” out of runtime into either modeling or compile time.
- A complex set of XML descriptor files which are difficult to get correct by hand as there are many rules that govern them above and beyond being well formed
- No formal meta-model or UML profile exists for the SCA
- While the SCA definitely raises the level of abstraction with regard to radio component development, it does not inherently provide an automatic and configurable means to get back to the lower, executable levels of abstraction or to its declarative languages.

3. ENTER DOMAIN-SPECIFIC MODELING

In order to tackle and tame the complexity of these systems and of the new specification it was necessary to provide:

- effective support under the SCA that allows users to program directly in the terms of the language of the domain and specification, ideally in graphical and declarative form to the greatest extent possible
- means to ensure that the programming is correct
- means to automatically generate executable 3GL programming language implementations from these models
- means to automatically generate additional software artifacts that are synchronized with the model

Those familiar with Domain-Specific Modeling will recognize the above bullets as part of the sacred triad of Domain-Specific Modeling: *Language*, *Editor*, and *Generator*. Couched in terms of Domain Specificity and at a finer granularity, these three elements map to:

- a *Domain-Specific Language* (DSL)
- a *Domain-Specific Graphical Language* and *Domain Specific Views* (DSGL, DSViews)
- a *Domain-Specific Constraint Language* (DSCL)
- a family of *Domain-Specific Code Generators* (DSG).

Table 1 lists the activities used in tackling the complexity in domain and then leveraging Domain Specific Modeling techniques to it

General Approach	Radio Domain
Isolate the abstractions and how they work together	*The SCA*
Create a formalized grammar for these – DSL	*Create a formalize SCA meta-model*
Create a graphical representation of the grammar – GDSL	*Create a SCA specific graphical tool*
Provide domain-specific constraints – GDSCL,DSCL	*Program into the tool the constraints*
Attach generators for necessary transformations	*C++, C, Ada and VHDL generators*

Table 1

One type of tool that can be used to develop the above software artifacts are what some refer to as Language Workbenches[2]; i.e. tools that allow a developer to define a domain-specific language and its graphical counter part, the editor, as well as a domain-specific generators that can iterate over the domain-specific model to produce executable artifacts. Some language workbenches available today include the Eclipse Modeling Framework and the Eclipse Graphical Editor Framework (EMF/GEF)[3], the Generic Modeling Environment (GME)[4], Microsoft's Visual Studio Team System Domain Specific Language Tools (VSTS DSL)[5], and MetaCase MetaEdit+ [8].

To allow users to run on multiple host platforms most easily and to integrate with addition eclipse tools and frameworks, we chose to use the EMF/GEF solution.

4. DEFINING THE DOMAIN-SPECIFIC LANGUAGE

The goal here is to provide a domain-specific higher level of abstraction with which both software and lay developers can program. Key to this is not only raising the level of abstraction but also providing domain-specific abstractions. Developers of SCA applications typically program in 3GL languages such as C, C++ and Ada. One of the goals of domain specific modeling is ***simplified modeling and programming in the problem space*** vs. ***complex modeling and programming in the solution space***. Figure 1 below juxtaposes two possible ways to represent the same concept in the SCA Software Defined Radio Domain. The left side diagram shows a typical UML diagram for a trivial SCA Component with two ports and two properties. The C++ source code is even more complicated. The right side diagram shows the same entity in terms of a higher abstract concept, a component with two ports and two properties, that is much more readable and less complex

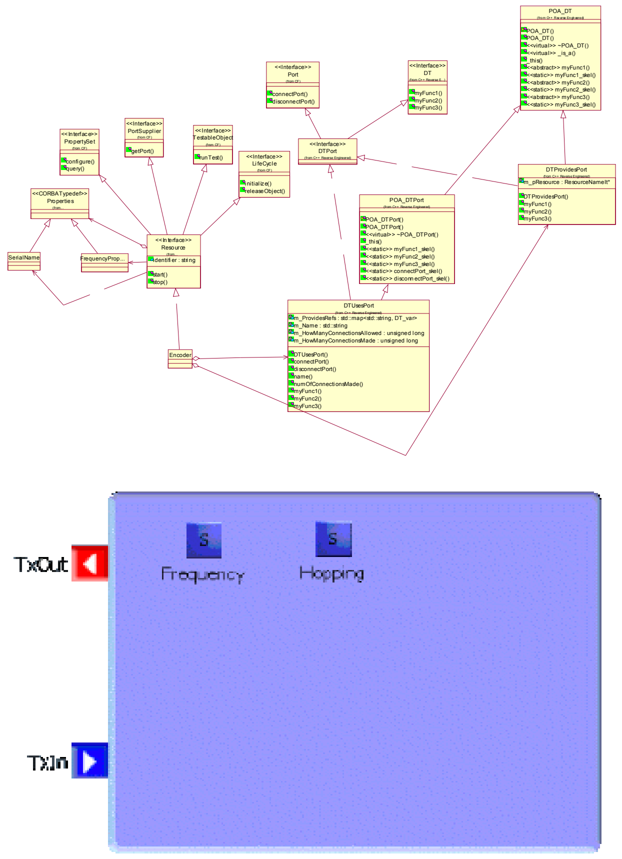


Figure 1

The raising of the level of abstraction is made possible through the creation of a formalized metamodel expressed in terms of the particular language workbench. In this case this involves creating a metamodel that the Eclipse Modeling Framework can understand. Fig 2 shows a greatly simplified metamodel for the SCA. Naturally, the full meta-model for the entire SCA is much more involved but for the purposes of demonstration and saving space we have presented a simplified version of it.

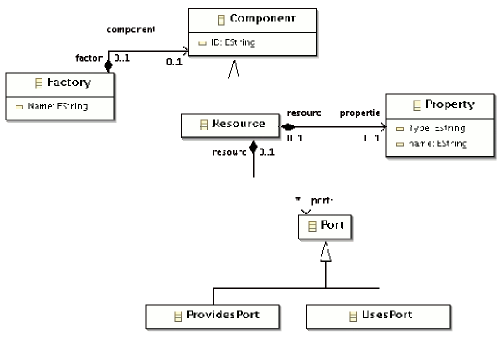


Figure 2

As stated before, the SCA provides a general architecture and UML diagrams as well as text-based behavioral descriptions and requirements and annotated XML DTD documents. While these are very detailed they are not formalized sufficiently to serve as a useful meta-model by themselves. The meta-model created and desribed here involved building upon the structure of the SCA and culling from the rest of the specification requirements, constraints and behaviors that together make up a complete and comprehensive meta-model characterizing the entire specification. As is usual, the group of developers building the meta-model are experienced SCA and software defined radio developers as well as experienced modelers.

It is from this meta-model that one provides the end user with the ability to program more directly in the domain. Additionally, end users are able to program more in the declarative than in the imperative; i.e. saying what they want to have, not specifying how it is to be done. Listing 1 shows a simple example of the persistent form of the Domain Specific Language in accordance with the metamodel.

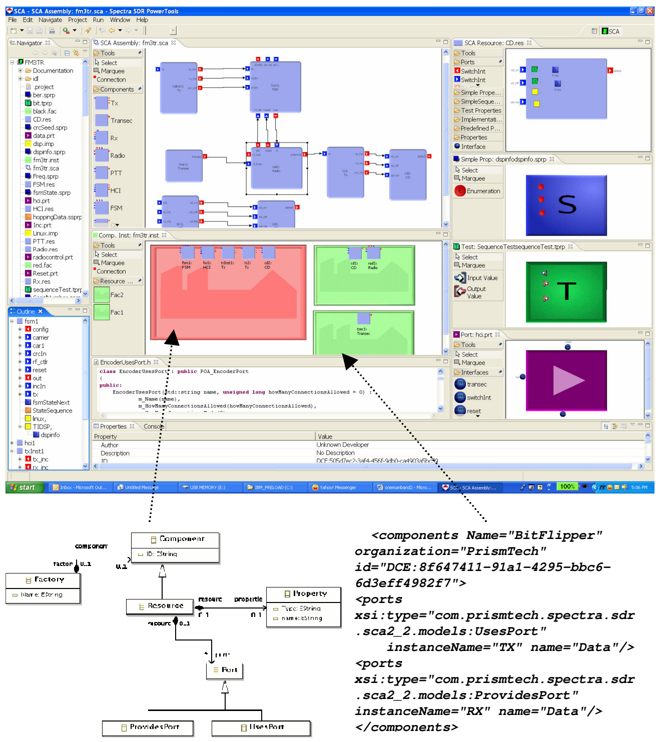
```
<?xml version="1.0" encoding="ASCII"?>
<com.prismtech.spectra.sdr.sca2_2.models:Asse
mbly
xmi:version="2.0"
xmlns:xmi="http://www.omg.org/XMI"
       xmlns:com.prismtech.spectra.sdr.sca2_2
.models="http://com.prismtech.spectra.sdr.sca
2_2.models">
  <components Name="BitFlipper"
organization="PrismTech" id="DCE:8f647411-
91a1-4295-bbc6-6d3eff4982f7">
       <ports
xsi:type="com.prismtech.spectra.sdr.sca2_2.mo
dels:UsesPort" instanceName="TX"
name="Data"/>
       <ports
xsi:type="com.prismtech.spectra.sdr.sca2_2.mo
dels:ProvidesPort"
instanceName="RX" name="Data"/>
  </components>
</com.prismtech.spectra.sdr.sca2_2.models:Ass
embly>
```

Listing 1

While providing a higher level of abstraction this text based language can still be labor intensive, error prone and hard to read. This leads directly into the next step of Domain-Specific Modeling.

5. DEFINING THE DOMAIN-SPECIFIC GRAPHICAL LANGUAGE (DSGL) AND VIEWS (DSV)

What is needed next is a way to express the Domain Specific Language graphically or visually. This involves working within your Language Workbench of choice to adorn the Domain-Specific Language with graphical and visual artifacts that allow the user to program quickly and correctly and in a way that communicates correctly the essence of the architecture and design.



```
  <components Name="BitFlipper"
organization="PrismTech"
id="DCE:8f647411-91a1-4295-bbc6-
6d3eff4982f7">
<ports
xsi:type="com.prismtech.spectra.sdr
.sca2_2.models:UsesPort"
    instanceName="TX" name="Data"/>
<ports
xsi:type="com.prismtech.spectra.sdr
.sca2_2.models:ProvidesPort"
instanceName="RX" name="Data"/>
</components>
```

Figure 3

Figure 3 shows the PrismTech Spectra SDR PowerTool modeling tool. This modeling tool allows end users to quickly and acurrately build software defined radio components and connect them together. The DSGL is built and based on the underlying meta-model described earlier and can be persisted in textual form for processing by other programs. It is through this DSGL that end users program with very intuitive icons, images, tools, artifacts and property sheets. Just as UML provides different views to describe various aspects of object-oriented systems so to does this tool provide Domain Specific Views that allow users to design, express and communicate domain specific aspects of their designs. Additionally, the Domain-Specific Modeling tool provides the end user with ability to program in the declarative versus the imperative.

6. THE DOMAIN-SPECIFIC CONSTRAINT LANGUAGE (DSCL)

Almost as important as what you see in the graphical tool illustrated in Figure 3 is what you don't see. The very fact that the DSGL is based on the meta-model means that it restricts programming to within the bounds of the meta-model. In other words, the tool is meta-model-centric as opposed to GUI-centric. In this case, the GUI itself forces the user to abide by the structural and creational aspects of the meta-model. This goes extremely far in allowing the developer to program quickly and correctly in terms of their domain. Additional constraints can be added via various programming facilities of the language workbench being used. Concrete SCA-unique examples of these types of constraints include not being able to connect ports that support different interfaces or not exceeding connection thresholds of output ports. These are errors that are typically allowed to creep into the runtime system which lead to expensive integration and support problems. By “left shifting” these potential defects into the modeling/compilation phase, we can simultaneously harness the dynamic nature of the SCA runtime component deployment, configuration and connection paradigm and do so in a correct and robust fashion. The DSCL enforces structural compositional, directional, etc. constraints, pre-conditions, post-conditions and invariants

7. DOMAIN-SPECIFIC GENERATORS

Ultimately, the tool must be able to transform the domain specific language into an executable or imperative format, or to a form that can be transform easily by other compilers into an executable form. This is achieved through the connection of *Domain Specific Generators* to the Domain Specific Editors. Embedded systems are frequently targeted at disparate processing elements (e.g. general-purpose processors, digital signal processors, field programmable gate arrays (FPGA)) and as such the tool needs to be able plug in multiple domain specific code generators that can iterate over the model and produce multiple types of executable code.

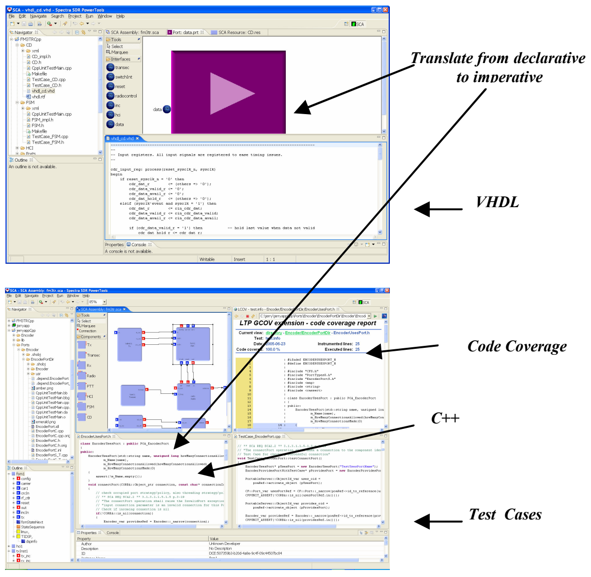


Figure 4

Figure 4 shows examples of the software artifacts coming from the domain-specific generators. Having the key information captured in the model, changes in the model are instantly reflected in the generated code.

The SCA architecture is most effectively implemented using a number of industry standard Design Patterns. Most notably are the Extension Object Pattern[6], Extension Interface Pattern[7] and the Component Configurator Pattern[7]. These patterns are typically repeated over and over again in an SCA implementation with minor paramaterization to account for the context in which they are used. The pre-validated implementations of these patterns can be generated directly from the domain specific generators. Many of these patterns capture infrastructure scaffolding, behavior required by the SCA specification as well as middleware concerns that can be difficult for radio developers to understand and get correct. Additional artifacts are generated from the model including, the XML descriptors, Unit Test Cases, documentation etc. The constraints of the tool straddle the editor and the generators. By using the generated code, the users can rely on prevalidated logic and patterns written by experts in the domain and thus they are “constrained”, if you will, to being correct in their implementation.

8. BENEFITS OF DOMAIN-SPECIFIC MODELING AS APPLIED TO SOFTWARE DEFINED RADIOS

A number of notable benefits become extremely apparent as a result of providing a domain modeling tool and all its constituent parts to the software defined radio domain.

- Increased productivity – users can program at a much higher level of abstraction and use generators to automatically get to lower levels that can thereafter be transformed and executed. The increased level of abstraction is coupled with the fact that the DSL is much more declarative in nature and so the users become less concerned with how actions are done and more concerned with *that* they are done. Users of the tool report a minimum of 500% increases in productivity and compare the magnitude of gains to be analogous to using a compiler to generate assembly code from higher order languages.
- Increased correctness – the generators provides prevalidated logic and other artifacts
- Synchronization of software artifacts. Since the artifacts are generated directly from the model, the maintenance burden of maintaining them all is greatly reduced
- Involvement of lay programmers and increased communication amongst company teams. Since the model is expressed in problem domain terms and

not solution domain terms, the communication of the model encompasses more disciplines beyond software engineering to include hardware and systems engineering and management teams.

- Lower cost of entry. As much of the infrastructure detail is captured in the metamodel, editor and generators, the learning curve of developing software defined radios for a particular domain is greatly reduced.
- Architectural consistency at the implementation level. While the SCA mandates architectural form at the interface level it does not at the implementation level. This opens the door to many different architectural implementations. While this is necessary in some uses cases, in many it is not and results in unnecessary complexity and maintenance burdens. The degree to which the applications have architectural consistency in their implementations determines the ease of maintenance by a central maintenance body.
- "Left shifting" of defects from runtime to modeling time. This provides orders of magnitude of cost savings across the development cycle

9. SUMMARY AND CONCLUSION

The history of software has seen the continued process of raising the level of programming abstraction while simultaneously providing an automatic and configurable means to traverse to lower levels of more executable forms of programs. Additionally, this evolution has included the continued introduction of ways and means to express domain concepts effectively so that the end user can program more directly in the problem space and not in the solution space. Using Model Driven Development and Domain-Specific Modeling via existing Language Workbenches is another effective step in this direction. Application of these techniques to the Software Radio Domain has yielded orders of magnitude of increase in productivity, correctness and robustness of these systems and can serve as the foundation for a graceful evolution of its products.

10. REFERENCES

[1]http://jtrs.army.mil/sections/technicalinformation/fset_technical_sca.html

[2]http://www.martinfowler.com/articles/languageWorkbench.html

[3] http://www.eclipse.org

[4] http://www.isis.vanderbilt.edu/Projects/gme/

[5]http://msmvps.com/vstsblog/archive/2005/07/02/56408.aspx

[6] http://www.smallmemory.com/almanac/Gamma98.html

[7] http://www.cs.wustl.edu/~schmidt/POSA/

[8] http://www.metacase.com/

[9] http://www.objectmentor.com/resources/articles/ocp.pdf

[10] http://www.aw-bc.com/catalog/academic/product/0,1144,0321278658,00.html

[11] http://www.aw-bc.com/catalog/academic/product/0,1144,0321146530,00.html

[12] http://www.aw-bc.com/catalog/academic/product/0,1144,0201485672,00.html

[13] http://www.aw-bc.com/catalog/academic/product/0,1144,0321213351,00.html

[14] http://web.it.kth.se/~jmitola/

Productivity Tools & Middleware

PRISMTECH

Using Domain-Specific Modeling and Model Driven Development to Develop Software Defined Radio Components and Applications

SDR '05
2005 Software Defined Radio Technical Conference and Product Exposition
Orange County, CA USA Nov 14-18, 2005

Dominick Paniscotti
Bruce Trask
Angel Roman
Vikram Bhanot

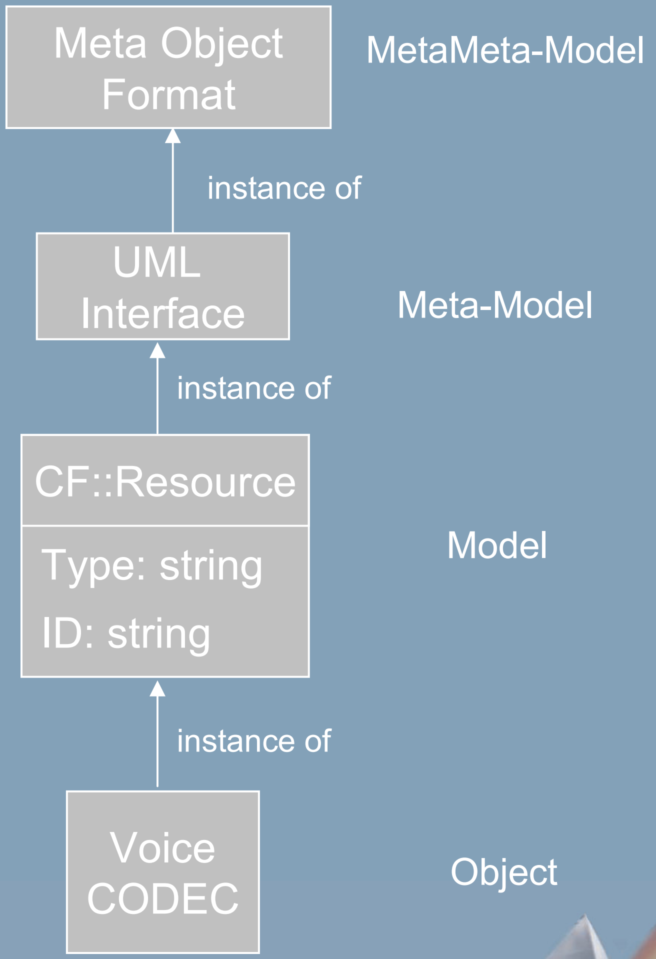
Spectra Software Defined Radio Products

What is a Domain Specific Language?

- In order to understand DSLs, one must understand levels of modeling
- DSLs are defined using Meta-Models
- Meta-Models are defined using even higher level models

So, a Domain Specific Language is…

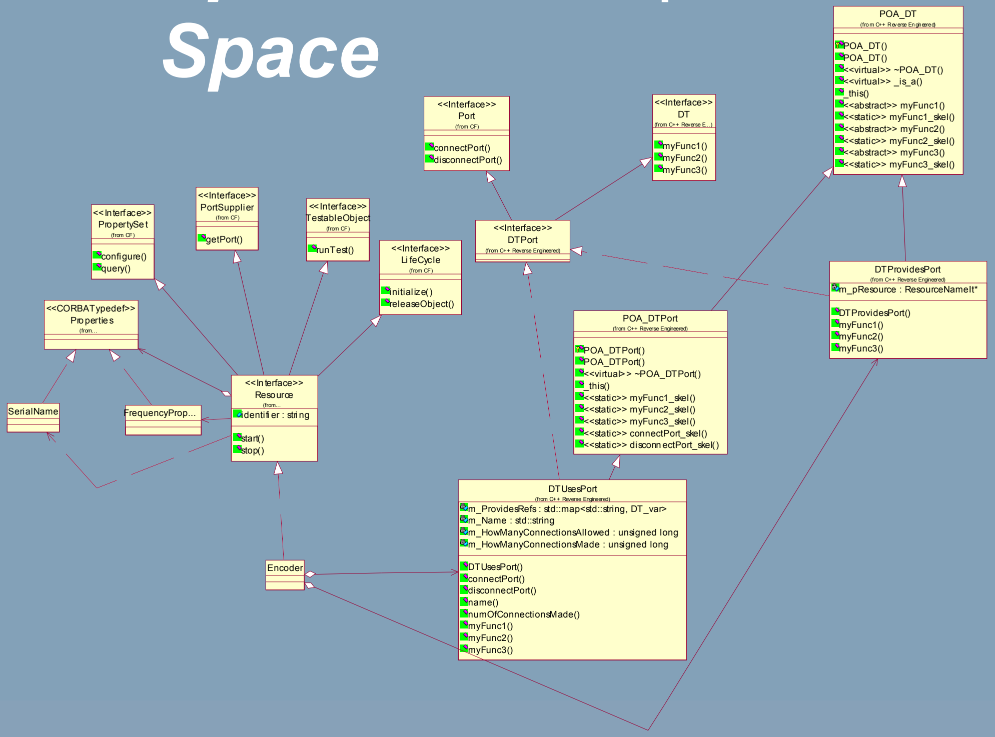
- A language targeted to a particular problem
  - Such as Software Radios
- Not a general purpose language aimed at any kind of problem
  - Such as UML

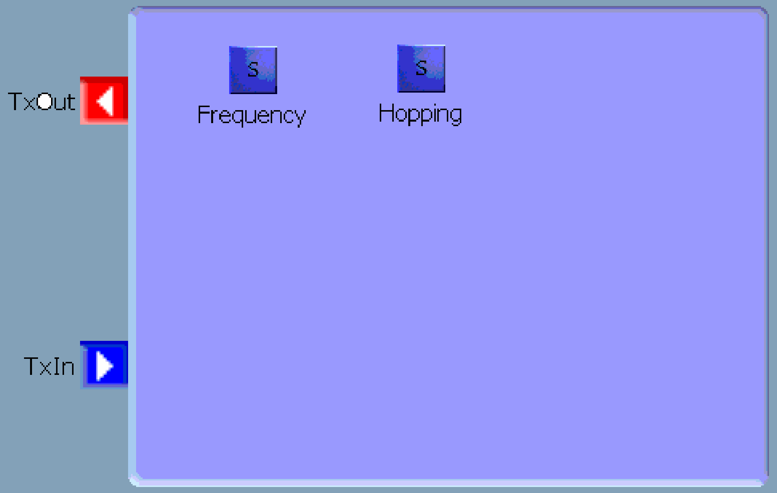


PRISMTECH

Domain Specific Modeling

- DSLs allow simplified modeling in the ***Problem Space*** vs. complex modeling ***in the Solution Space***

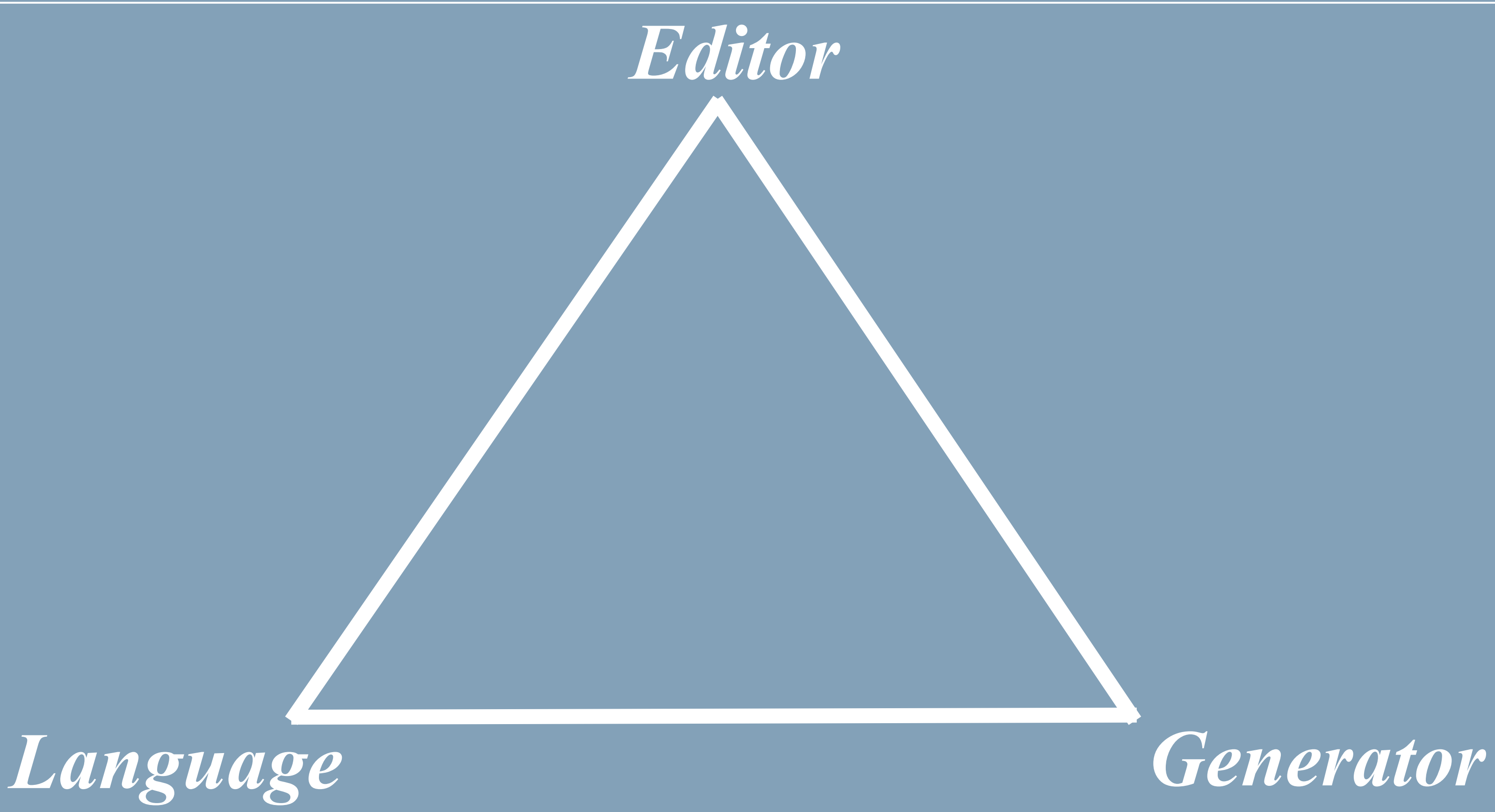




Solution Space Modeling *Problem Space Modeling*

Single CF::Resource with 2 Port and 2 Properties

Domain Specific Modeling



Model Driven Software Development

- ***Model Driven Development***
  - Models are used to express the design of a system
  - Models are transformed to create implementations
- Model Driven approaches combined with ***Generative Programming*** technologies allow developers to ***"create designs that write code"***
- Developers use ***Domain Specific Languages*** and ***Models*** to ***efficiently and automatically map to platform-specific technologies***
- Model Driven approaches allow us to effectively and practically move from making one-time concrete systems to ***families of systems***
  - Since the meta-models capture the system family rules

Model Driven Software Development

- Allows developers to ***weave various aspects*** of the solution domain together automatically
- Increases ***productivity and correctness*** in complex systems by ***simplifying*** development
- Puts tools in the hands of developers so they can properly ***capture the commonalities and variabilities*** of their domain
- Captures the "***sweet-spot" of many areas of software development***
  - Including modeling, code generation, coding, testing
  - Could program in assembly or C++, what is the best combination of software tools to get the job done and done correctly
- Supports the creation of development ***processes*** for the domain
  - Explain exactly what is meant here
- Supports the creation of ***Domain Specific Tools***
- These Tools further ***eliminate the complexities associated with development*** in a particular domain

PRISMTECH

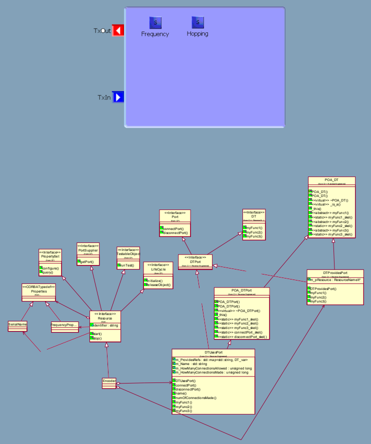
Levels of Abstraction

Less Flexible

Flexibility

More Flexible

Programming



Higher Order Languages

Assembly

Opcodes

Custom Hardware

Platform

Middleware

Operating Systems

More Productive

Productivity

Less Productive

Generative Programming

- The process of moving from a higher level abstraction to a lower level abstraction *automatically*
- Specification of transformation rules support this paradigm

  e.g. C++ to Assembly to Opcodes.
- Domain Specific Models and Languages work in concert with generative technologies
  - The whole is the worth more than the sum of the parts
- Thereby increasing the productivity

PRISMTECH

To summarize…

- Model driven development defines higher levels of domain specific abstractions and combines these with generators that automatically transform these abstractions to lower level ***executable*** artifacts
- An Engineering Trade-Off
  - Sacrificing flexibility for productivity
- Not a value judgment
  - Its payback is in the cost reduction found in developing ***Families Of Systems*** vs. Single Concrete System

PRISMTECH

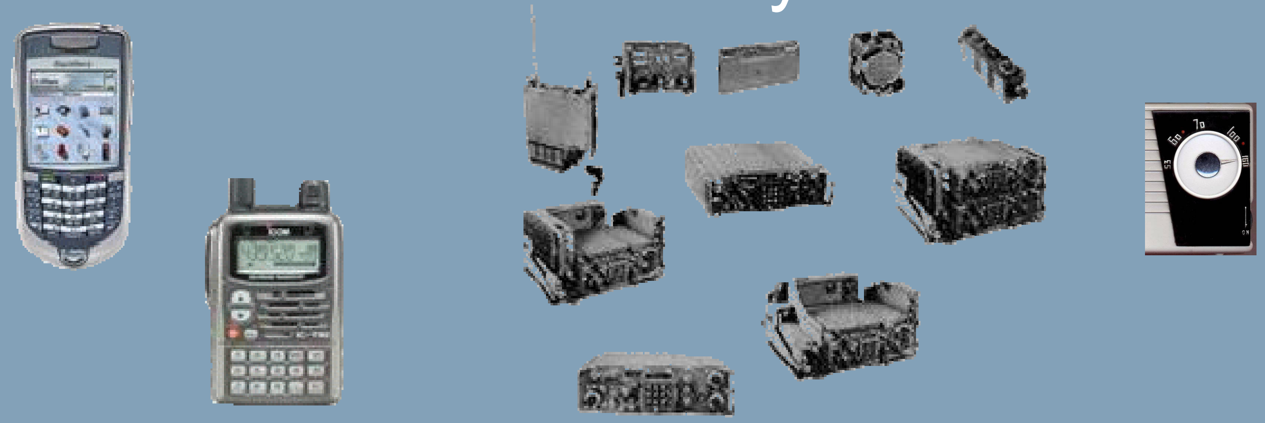
Families Of Systems

- Focus on System Families allows one to identify the commonalities and variablities found across family members
- And develop DSLs to:
  - ***Factor out*** common behavior into paramaterizable abstractions
  - Provide ***extension mechanisms*** to incorporate variation points found across family members
- And further develop generators to synthesize concrete functionality for a particular family member

PRISMTECH

Radio Families

Radio Family Members



- ***Commonalities***
  - Properties
  - Tests
  - Life Cycle
  - Communications Path
  - Deployment
  - Functionality (Routing, Networking)
  - Basic architecture
- ***Variabilities***
  - Functionality
  - RF or SiS characteristics
  - Processing Elements (HW)
  - Size weight and power constraints

What the SCA has done … and has not

- SCA isolated the commonalities and variabilities but did not provide a DSL in which to program these things
- No generators
- So to really complete the picture, need generators to handle practical use and to map against the variabilities
- The SCA provides the necessary abstractions and framework of patterns (Extension Object with component Configurator) as well as the deployment and configuration engine.
- What is needed now are the remaining artifacts to make this particular solution complete.

PRISMTECH

Providing the remaining steps

- Allow programmers to program in the higher order domain by
  - Providing a domain specific grammar
  - Graphical representations of this grammar
  - Automatic constraint engines to ensure the use of the grammar is correct
  - Automatic generations engines
    - transform the resulting model to various targets along varying dimensions
    - Weave together various complex aspects of the domain

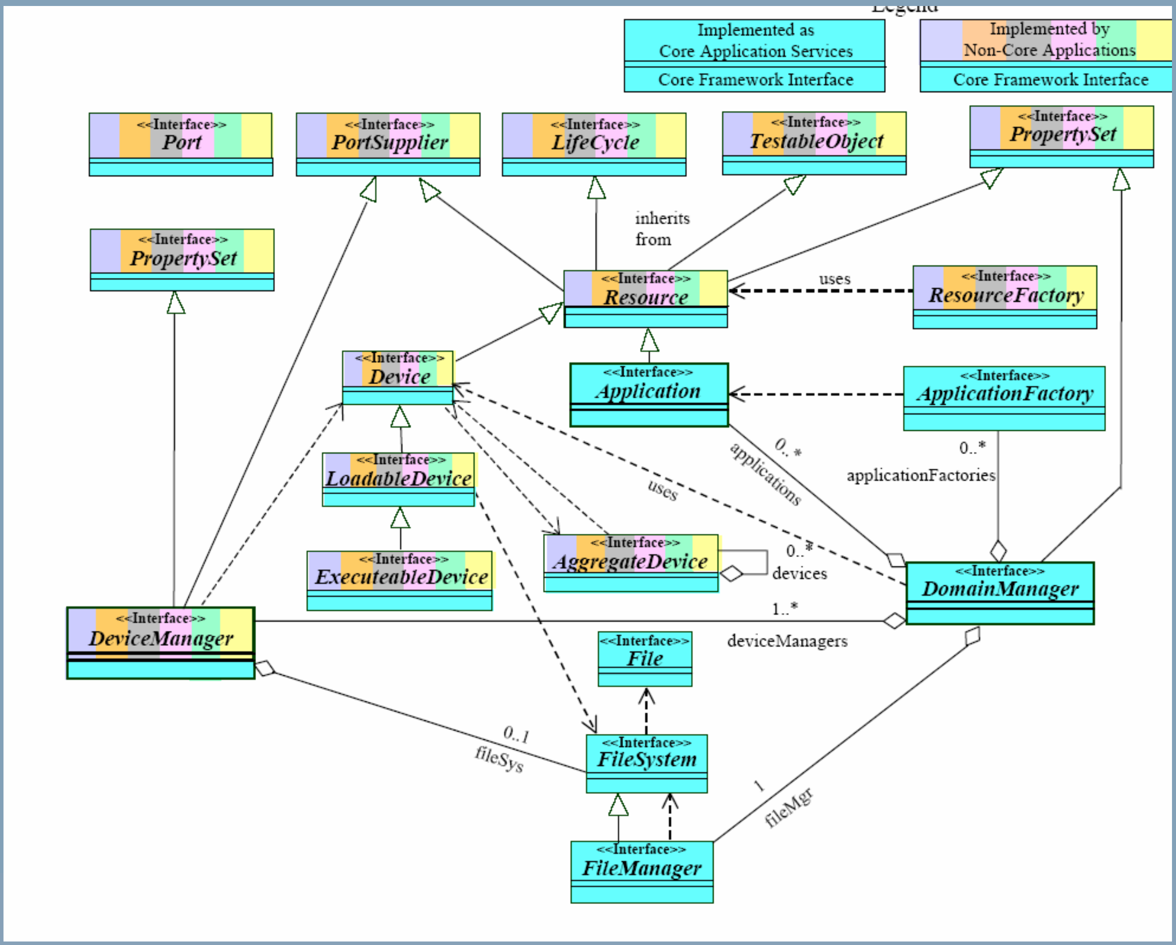
PRISMTECH

The steps

In general	In our domain
Isolate the abstractions and how they work together	*The SCA*
Create a formalized grammar for these - DSL	*Create a formalize SCA meta-model*
Create a graphical representation of the grammar – GDSL	*Create a SCA specific graphical tool*
Provide domain-specific constraints – GDSCL, DSCL	*Program into the tool the constraints*
Attach generators for necessary transformations	*C++, C and VHDL generators*

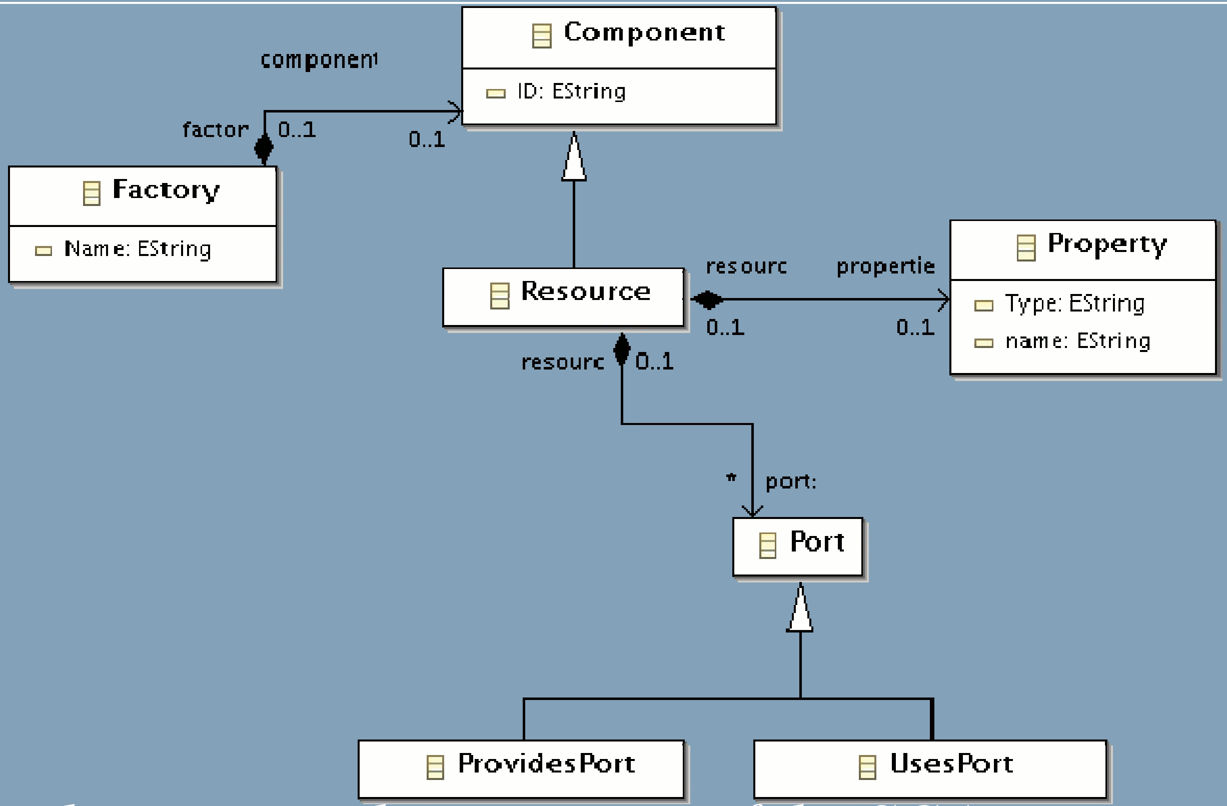
PRISMTECH

The SCA


Legend
Implemented as Core Application Services
Core Framework Interface
Implemented by Non-Core Applications
Core Framework Interface
<<Interface>> Port
<<Interface>> PortSupplier
<<Interface>> LifeCycle
<<Interface>> TestableObject
<<Interface>> PropertySet
<<Interface>> PropertySet
inherits from
<<Interface>> Resource
uses
<<Interface>> ResourceFactory
<<Interface>> Device
<<Interface>> Application
<<Interface>> ApplicationFactory
0..*
applications
0..*
applicationFactories
<<Interface>> LoadableDevice
uses
<<Interface>> AggregateDevice
0..*
devices
<<Interface>> ExecuteableDevice
<<Interface>> DomainManager
<<Interface>> DeviceManager
1..*
deviceManagers
<<Interface>> File
0..1
fileSys
<<Interface>> FileSystem
1
fileMgr
<<Interface>> FileManager

PRISMTECH

The Metamodel



In addition to the structural constraints of the SCA, we can

- *further expand the meta-model to include the many additional requirements and constraints within the SCA*

- *constraints that assist in locating errors at modeling time vs. runtime*

PRISMTECH

Domain Specific Language

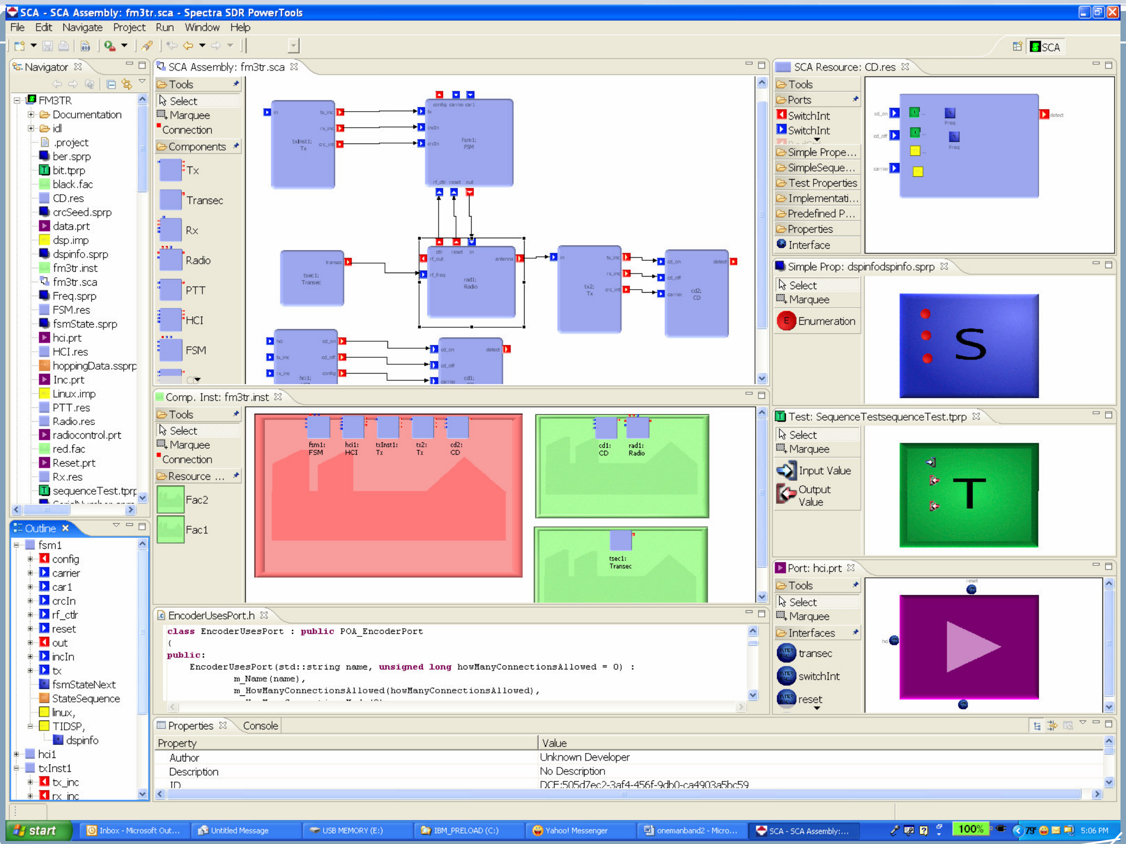
```
<?xml version="1.0" encoding="ASCII"?>
<com.prismtech.spectra.sdr.sca2_2.models:Assembly
          xmi:version="2.0"
          xmlns:xmi="http://www.omg.org/XMI"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xmlns:com.prismtech.spectra.sdr.sca2_2.models="http://com.prismtech.spectra.sdr.sca
2_2.models">
<components Name="BitFlipper" organization="PrismTech" id="DCE:8f647411-91a1-4295-bbc6-
6d3eff4982f7">
    <ports xsi:type="com.prismtech.spectra.sdr.sca2_2.models:UsesPort"
                                            instanceName="TX" name="Data"/>
    <ports xsi:type="com.prismtech.spectra.sdr.sca2_2.models:ProvidesPort" instanceName="RX"
name="Data"/>

  </components>
</com.prismtech.spectra.sdr.sca2_2.models:Assembly>
```

PRISMTECH

A Graphical Domain-Specific Language

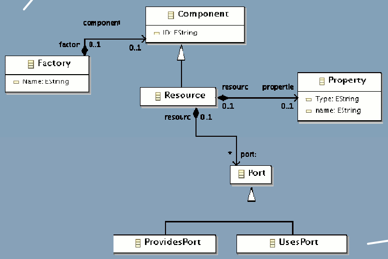
Images, layout, organization based on meta-model



EncoderUsesPort.h

```
class EncoderUsesPort : public POA_EncoderPort
{
public:
    EncoderUsesPort(std::string name, unsigned long howManyConnectionsAllowed = 0) :
            m_Name(name),
            m_HowManyConnectionsAllowed(howManyConnectionsAllowed),
```

Property	Value
Author	Unknown Developer
Description	No Description
ID	DCE:505d7ec2-3af4-456f-9db0-ca4903a5bc59



```
    <components Name="BitFlipper" organization="PrismTech"
id="DCE:8f647411-91a1-4295-bbc6-6d3eff4982f7">
<ports xsi:type="com.prismtech.spectra.sdr.sca2_2.models:UsesPort"
    instanceName="TX" name="Data"/>
<ports xsi:type="com.prismtech.spectra.sdr.sca2_2.models:ProvidesPort"
instanceName="RX" name="Data"/>
</components>
</com.prismtech.spectra.sdr.sca2_2.models:Assembly>
```

Declarative Specification/Programming

- Both the DSL and GDSL are *declarative* in nature
- Involves programming by setting properties, making connections and establishing relationships
  - Versus imperative sequential procedural instructions[1]
- "Declarative programming improves productivity and quality because it is another form of reuse of preprogrammed, prevalidated logic"[1]

[1] *Model Driven Architecture D. Frankel, Wiley 2003*

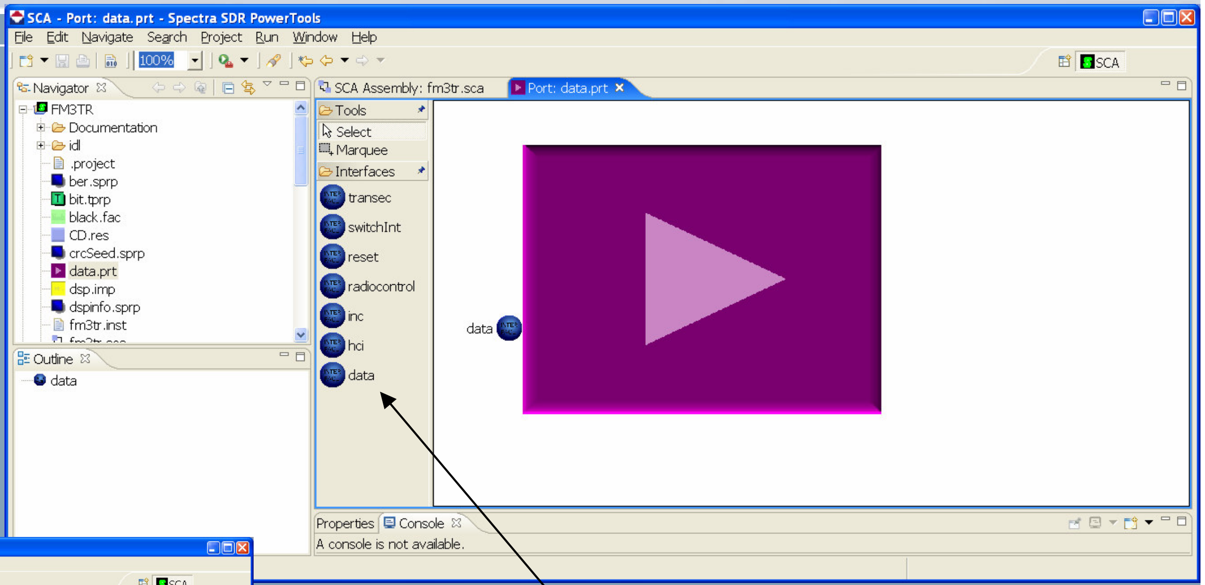
PRISMTECH

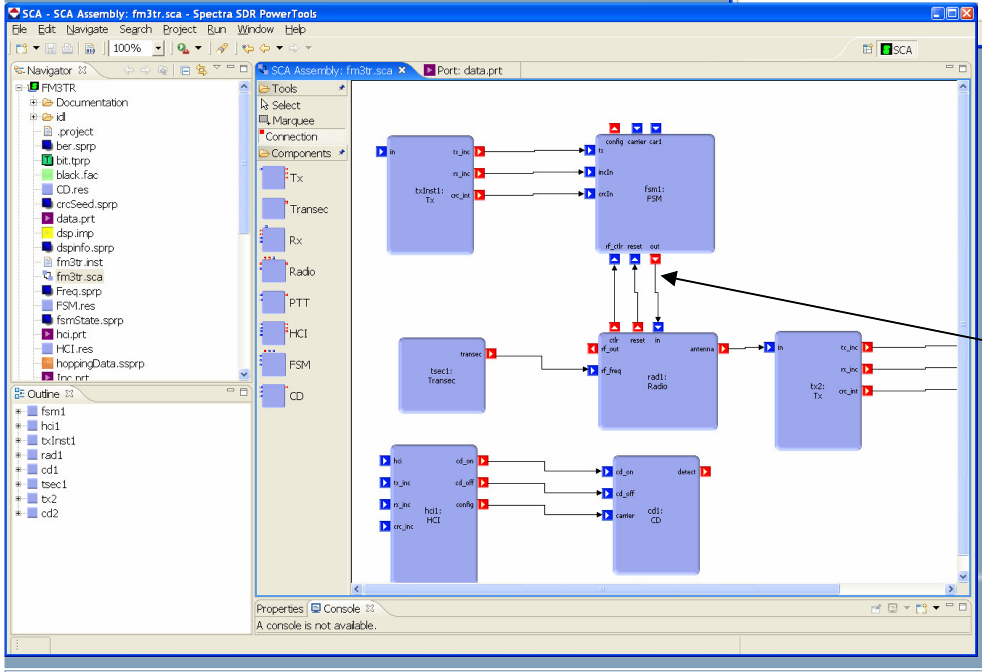
Processing the Declarative Specification

- Constraining its input
- Interpreting it as it is entered
- After interpretation, have the declarative specification drive code generators which will transform the model into an executable form for a given platform

PRISMTECH

Domain-Specific Constraints





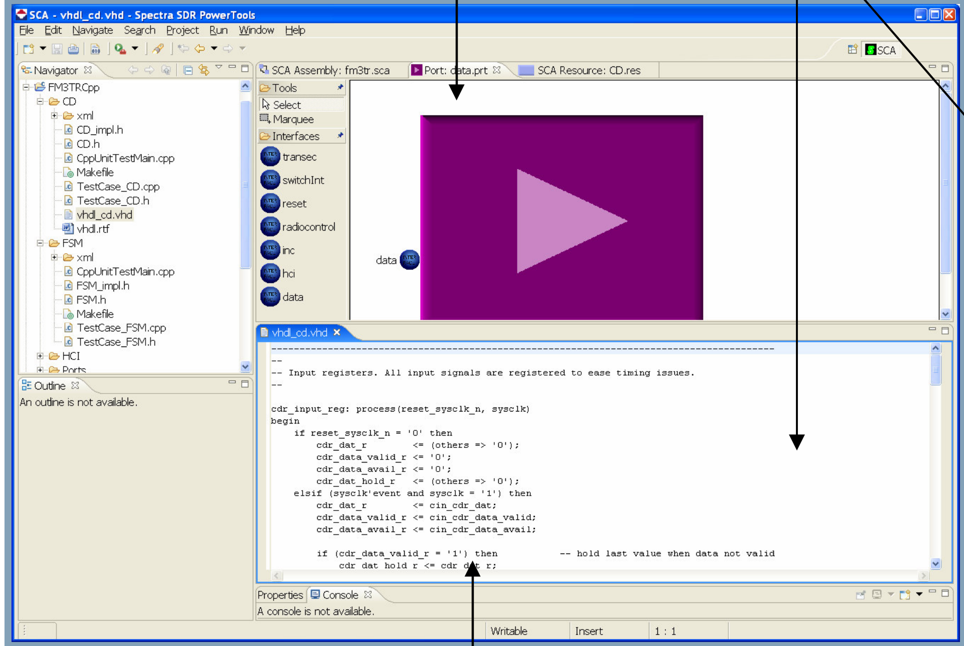
Enforce structural compositional, directional, etc constraints. Preconditions, postconditions and invariants

PRISMTECH

Generators

Translate from declarative to imperative

Code Coverage



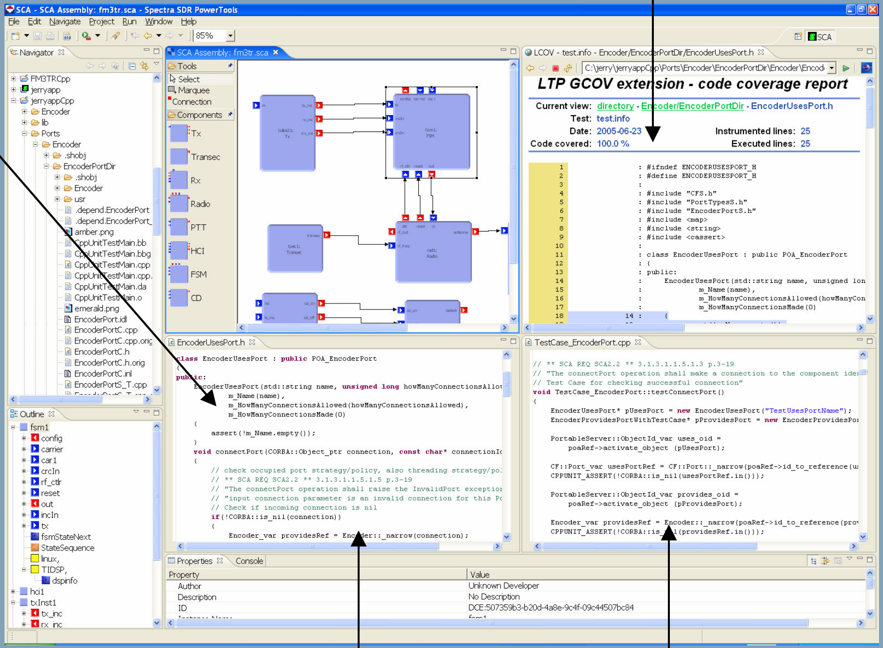
vhdl_cd.vhd

```
-------------------------------------------------------------------------------
--
-- Input registers. All input signals are registered to ease timing issues.
--

cdr_input_reg: process(reset_sysclk_n, sysclk)
begin
    if reset_sysclk_n = '0' then
        cdr_dat_r         <= (others => '0');
        cdr_data_valid_r <= '0';
        cdr_data_avail_r <= '0';
        cdr_dat_hold_r   <= (others => '0');
    elsif (sysclk'event and sysclk = '1') then
        cdr_dat_r         <= cin_cdr_dat;
        cdr_data_valid_r <= cin_cdr_data_valid;
        cdr_data_avail_r <= cin_cdr_data_avail;

        if (cdr_data_valid_r = '1') then           -- hold last value when data not valid
            cdr_dat_hold_r <= cdr_dat_r;
```

A console is not available.



LTP GCOV extension - code coverage report

Current view: directory - Encoder/EncoderPortDir - EncoderUsesPort.h
Test: test.info
Date: 2005-06-23
Code covered: 100.0 %
Instrumented lines: 25
Executed lines: 25

```
   : #ifndef ENCODERUSESPORT_H
   : #define ENCODERUSESPORT_H
   :
   : #include "CFS.h"
   : #include "PortTypesS.h"
   : #include "EncoderPortS.h"
   : #include <map>
   : #include <string>
   : #include <cassert>
   :
   : class EncoderUsesPort : public POA_EncoderPort
   : {
   : public:
   :     EncoderUsesPort(std::string name, unsigned lon
   :             m_Name(name),
   :             m_HowManyConnectionsAllowed(howManyCon
   :             m_HowManyConnectionsMade(0)
14 :     {
```

EncoderUsesPort.h

```
class EncoderUsesPort : public POA_EncoderPort
{
public:
    EncoderUsesPort(std::string name, unsigned long howManyConnectionsAllo
            m_Name(name),
            m_HowManyConnectionsAllowed(howManyConnectionsAllowed),
            m_HowManyConnectionsMade(0)
    {
        assert(!m_Name.empty());
    }
    void connectPort(CORBA::Object_ptr connection, const char* connectionId
    {
        // check occupied port strategy/policy, also threading strategy/po
        // ** SCA REQ SCA2.2 ** 3.1.3.1.1.5.1.5 p.3-19
        // "The connectPort operation shall raise the InvalidPort exceptio
        // "input connection parameter is an invalid connection for this Po
        // Check if incoming connection is nil
        if(!CORBA::is_nil(connection))
        {
            Encoder_var providesRef = Encoder::_narrow(connection);
```

TestCase_EncoderPort.cpp

```
// ** SCA REQ SCA2.2 ** 3.1.3.1.1.5.1.3 p.3-19
// "The connectPort operation shall make a connection to the component ide
// Test Case for checking successful connection"
void TestCase_EncoderPort::testConnectPort()
{
    EncoderUsesPort* pUsesPort = new EncoderUsesPort("TestUsesPortName");
    EncoderProvidesPortWithTestCase* pProvidesPort = new EncoderProvidesPo

    PortableServer::ObjectId_var uses_oid =
        poaRef->activate_object (pUsesPort);

    CF::Port_var usesPortRef = CF::Port::_narrow(poaRef->id_to_reference(u
    CPPUNIT_ASSERT(!CORBA::is_nil(usesPortRef.in()));

    PortableServer::ObjectId_var provides_oid =
        poaRef->activate_object (pProvidesPort);

    Encoder_var providesRef = Encoder::_narrow(poaRef->id_to_reference(pro
    CPPUNIT_ASSERT(!CORBA::is_nil(providesRef.in()));
```

Property	Value
Author	Unknown Developer
Description	No Description
ID	DCE:507359b3-b20d-4a8e-9c4f-09c44507bc84

VHDL

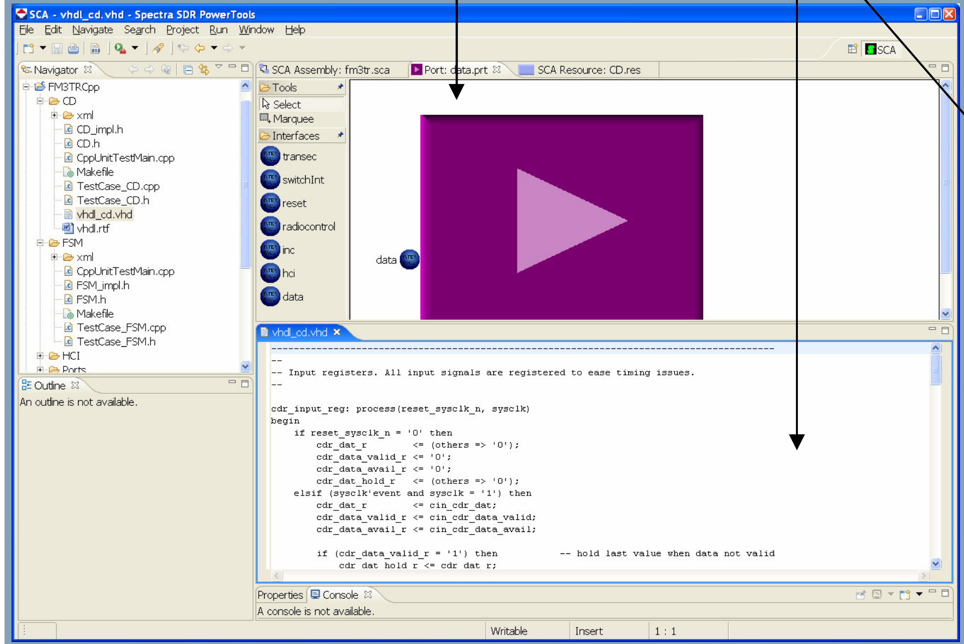
C++

Test Cases

PRISMTECH

Generators, continued

Application of Design Patterns Automatically



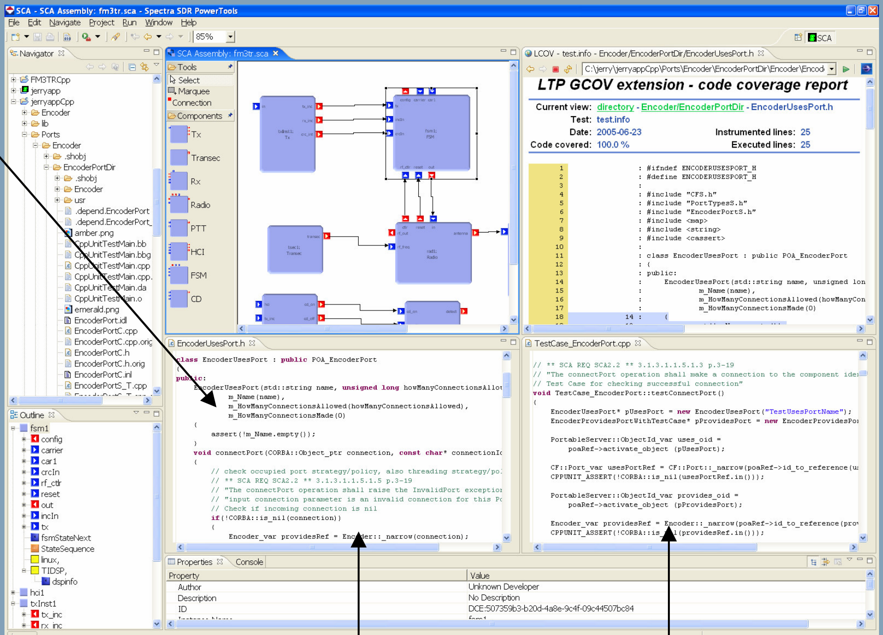
An outline is not available.

```
--------------------------------------------------------------------------------
--
-- Input registers. All input signals are registered to ease timing issues.
--

cdr_input_reg: process(reset_sysclk_n, sysclk)
begin
    if reset_sysclk_n = '0' then
        cdr_dat_r        <= (others => '0');
        cdr_data_valid_r <= '0';
        cdr_data_avail_r <= '0';
        cdr_dat_hold_r   <= (others => '0');
    elsif (sysclk'event and sysclk = '1') then
        cdr_dat_r        <= cin_cdr_dat;
        cdr_data_valid_r <= cin_cdr_data_valid;
        cdr_data_avail_r <= cin_cdr_data_avail;

        if (cdr_data_valid_r = '1') then           -- hold last value when data not valid
            cdr_dat_hold_r <= cdr_dat_r;
```

Properties | Console
A console is not available.
Writable | Insert | 1 : 1



LCOV - test.info - Encoder/EncoderPortDir/EncoderUsesPort.h

C:\jerry\jerryappCpp\Ports\Encoder\EncoderPortDir\Encoder\Encod

LTP GCOV extension - code coverage report

Current view: directory - Encoder/EncoderPortDir - EncoderUsesPort.h
Test: test.info
Date: 2005-06-23 Instrumented lines: 25
Code covered: 100.0 % Executed lines: 25

```
   : #ifndef ENCODERUSESPORT_H
   : #define ENCODERUSESPORT_H
   :
   : #include "CFS.h"
   : #include "PortTypesS.h"
   : #include "EncoderPortS.h"
   : #include <map>
   : #include <string>
   : #include <cassert>
   :
   : class EncoderUsesPort : public POA_EncoderPort
   : {
   : public:
   :     EncoderUsesPort(std::string name, unsigned lon
   :             m_Name(name),
   :             m_HowManyConnectionsAllowed(howManyCon
   :             m_HowManyConnectionsMade(0)
14 :     {
```

EncoderUsesPort.h

```
class EncoderUsesPort : public POA_EncoderPort
{
public:
    EncoderUsesPort(std::string name, unsigned long howManyConnectionsAllo
            m_Name(name),
            m_HowManyConnectionsAllowed(howManyConnectionsAllowed),
            m_HowManyConnectionsMade(0)
    {
        assert(!m_Name.empty());
    }
    void connectPort(CORBA::Object_ptr connection, const char* connectionI
    {
        // check occupied port strategy/policy, also threading strategy/po
        // ** SCA REQ SCA2.2 ** 3.1.3.1.1.5.1.5 p.3-19
        // "The connectPort operation shall raise the InvalidPort exceptio
        // "input connection parameter is an invalid connection for this P
        // Check if incoming connection is nil
        if(!CORBA::is_nil(connection))
        {
            Encoder_var providesRef = Encoder::_narrow(connection);
```

TestCase_EncoderPort.cpp

```
// ** SCA REQ SCA2.2 ** 3.1.3.1.1.5.1.3 p.3-19
// "The connectPort operation shall make a connection to the component ide
// Test Case for checking successful connection"
void TestCase_EncoderPort::testConnectPort()
{
    EncoderUsesPort* pUsesPort = new EncoderUsesPort("TestUsesPortName");
    EncoderProvidesPortWithTestCase* pProvidesPort = new EncoderProvidesPo

    PortableServer::ObjectId_var uses_oid =
        poaRef->activate_object (pUsesPort);

    CF::Port_var usesPortRef = CF::Port::_narrow(poaRef->id_to_reference(u
    CPPUNIT_ASSERT(!CORBA::is_nil(usesPortRef.in()));

    PortableServer::ObjectId_var provides_oid =
        poaRef->activate_object (pProvidesPort);

    Encoder_var providesRef = Encoder::_narrow(poaRef->id_to_reference(pro
    CPPUNIT_ASSERT(!CORBA::is_nil(providesRef.in()));
```

Property	Value
Author	Unknown Developer
Description	No Description
ID	DCE:507359b3-b20d-4a8e-9c4f-09c44507bc84

C++ *Test Cases*

- *Replicated production of Design Pattern(s) implementations*
- *Can be more effective than dealing with the replication using 3GL language features, particularly in DRE Systems*

PRISMTECH

The Benefits

- The SCA
  - portability
  - standardized development
- Meta-model – Domain Specific Language
  - more productivity
- GDSL
  - easy to use and communicate to others
- Constraints
  - left shift defects from run-time to modeling time
  - Harness the inherent flexibility provided by the SCA
- Generators
  - Productivity
  - Portability
  - Correctness
  - Architectural consistency
    - A coordinated set of design patterns[1]
  - Requirements traceability
  - Synchronization of software artifacts
    - e.g.documentation
  - Automated testing = increased robustness
  - Allow the marriage of design artifacts with development artifacts

[1] *Model Driven Architecture D. Frankel, Wiley 2003*

PRISMTECH

Contact Info

www.prismtech.com

Dominick - dp@prismtech.com

Bruce – bt@prismtech.com

PrismTech